Section 11: FFS, Virtual Memory

CS162

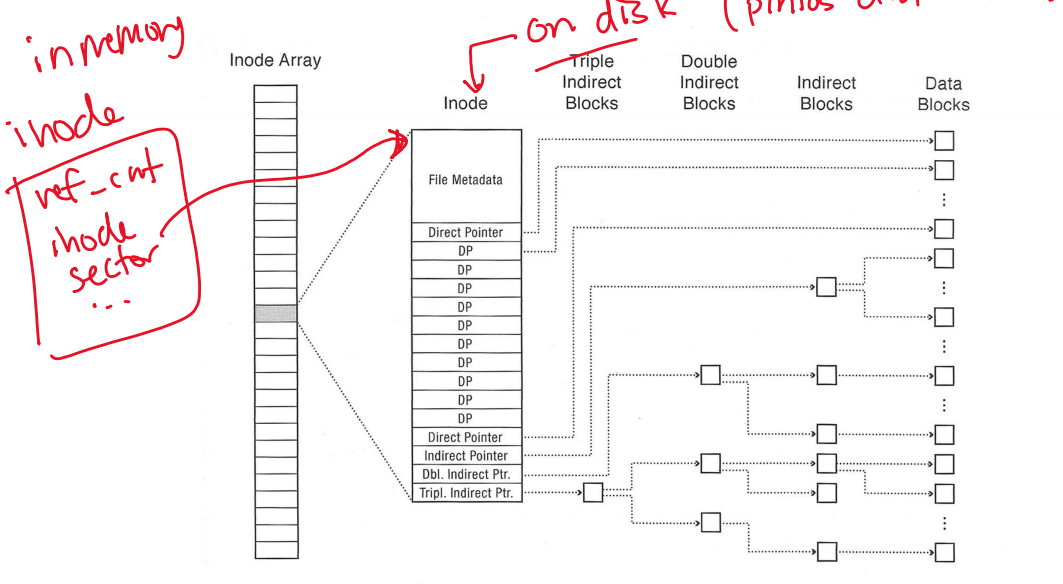
July 31, 2019

Contents

1 Vocabulary

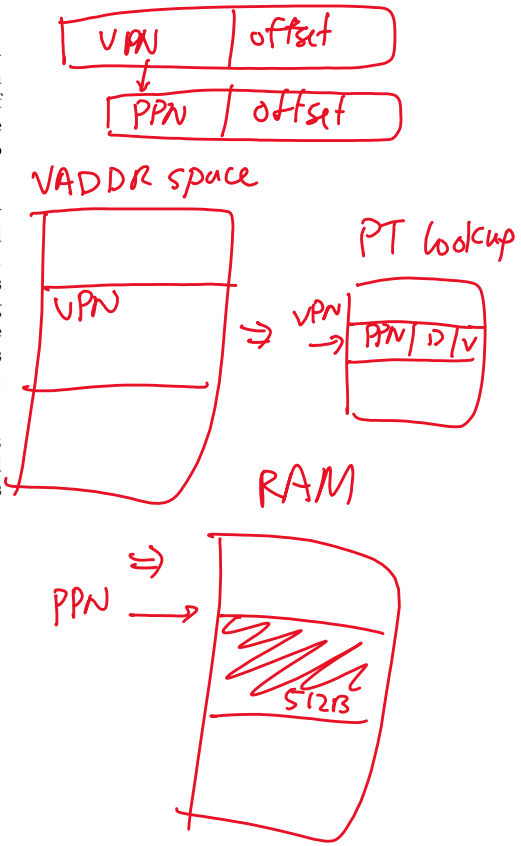
- **Unix File System (Fast File System)** - The Unix File System is a file system used by many Unix and Unix-like operating systems. Many modern operating systems use file systems that are based off of the Unix File System.
- **inode** - An inode is the data structure that describes the metadata of a file or directory. Each inode contains several metadata fields, including the owner, file size, modification time, file mode, and reference count. Each inode also contains several data block pointers, which help the file system locate the file's data blocks.

  Each inode typically has 12 direct block pointers, 1 singly indirect block pointer, 1 doubly indirect block pointer, and 1 triply indirect block pointer. Every direct block pointer directly points to a data block. The singly indirect block pointer points to a block of pointers, each of which points to a data block. The doubly indirect block pointer contains another level of indirection, and the triply indirect block pointer contains yet another level of indirection.



- **Virtual Memory** - Virtual Memory is a memory management technique in which every process operates in its own address space, under the assumption that it has the entire address space to itself. A virtual address requires translation into a physical address to actually access the system's memory.
- **Memory Management Unit** - The memory management unit (MMU) is responsible for translating a process' virtual addresses into the corresponding physical address for accessing physical memory. It does all the calculation associating with mapping virtual address to physical addresses, and then populates the address translation structures.

- **Address Translation Structures** - There are two kinds you learned about in lecture: segmentation and page tables. Segments are linearly addressed chunks of memory that typically contain logically-related information, such as program code, data, stack of a single process. They are of the form (s,i) where memory addresses must be within an offset of i from base segment s. A page table is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses.
- **Inverted Page Table** - The inverted page table scheme uses a page table that contains an entry for each phiscial frame, not for each logical page. This ensures that the table occupies a fixed fraction of memory. The size is proportional to physical memory, not the virtual address space. The inverted page table is a global structure – there is only one in the entire system. It stores reverse mappings for all processes. Each entry in the inverted table contains has a tag containing the task id and the virtual address for each page. These mappings are usually stored in associative memory (remember fully associative caches from 61C?). Associatively addressed memory compares input search data (tag) against a table of stored data, and returns the address of matching data. They can also use actual hash maps.
- **Translation Lookaside Buffer (TLB)** - A translation lookaside buffer (TLB) is a cache that memory management hardware uses to improve virtual address translation speed. It stores virtual address to physical address mappings, so that the MMU can store recently used address mappings instead of having to retrieve them mutliple times through page table accesses.



2 FFS Problems

2.1 File properties

Consider the following `inode_disk` struct, which is used on a disk with a 512 byte block size.

```
/* Definition of block_sector_t */
typedef uint32_t block_sector_t;

/* Contents of on-disk inode. Must be exactly 512 bytes long. */
struct inode_disk
  {
    off_t length;                       /* File size in bytes. */
    block_sector_t direct[12];          /* 12 direct pointers */
    block_sector_t indirect;            /* a singly indirect pointer */
    uint32_t unused[114];               /* Not used. */
  };
```

Why isn't the file name stored inside the `inode_disk` struct?

is not useful there. name associated w/ directory entry.

What is the maximum file size supported by this inode design?

$512 \cdot 12 + 128 \cdot 512 = 140 \cdot 2^9$ or $2^{16} + 12 \cdot 2^9$

How would you design the in-memory representation of the indirect block? (e.g. the disk sector that corresponds to an inode's `indirect` member)

if in powers of 2

block_sector_t [128], fits 128 dp's

2.2 Lookup

Sector $\lceil \frac{1000}{512} \rceil = 2$

Using the inode_disk defined above, how many disk sectors would you need to read to get the 1000^{th} byte of the file?

it's a dp, so 1 inode_disk and 1 data sector

2

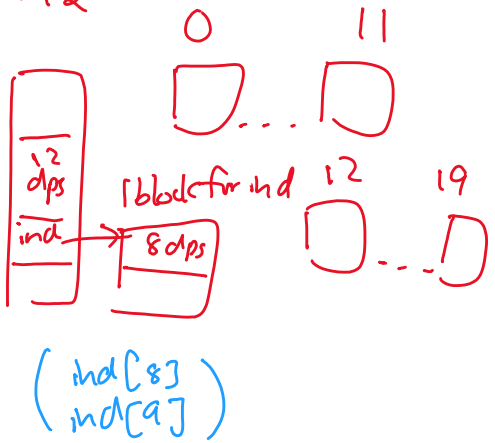
Using the inode_disk defined above, how many disk sectors would you need to read to get the $10,000^{th}$ byte of the file?

$\lceil \frac{10,000}{512} \rceil$

>12

indirect pointer, so 1 inode_disk, 1 indirect, 1 data

3



2.3 File Expansion

Lets say we had a file that was already 10240 bytes long, which equates to 20 disk sectors of data. Explain or draw the current state of the file. How many disk sectors have been allocated to store this file?

20 data, 1 ind, 1 inode_dis ⇒ 22

If we wanted to write 1024 bytes appended to the end of this file, how would we have to modify our state above?

allocate 2 disk sectors, update ind block, change filesize

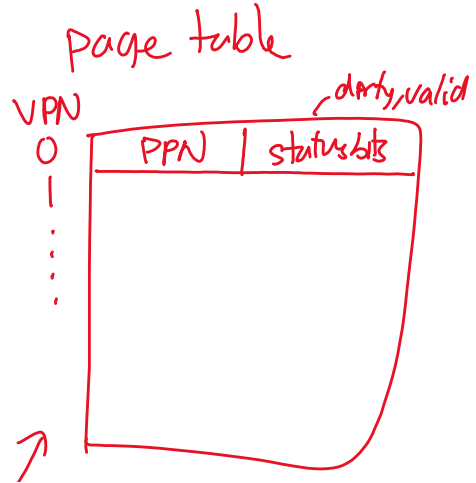
↑ entries 21-12 = 9, 22-12 = 10 (ind[8], ind[9])

3 Virtual Memory Problems

3.1 Conceptual Questions

If the physical memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

⇒ PPN is now 1 bit longer
⇒ each entry is 1 bit longer



If the physical memory size (in bytes) is doubled, how does the number of entries in the page map change?

virtual pages = # PTEs

same # entries, no change to VADDR space

If the virtual memory size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

does not change. same PPN length → no entry actually changes

If the virtual memory size (in bytes) is doubled, how does the number of entries in the page map change?

doubled since and twice as many pages

If the page size (in bytes) is doubled, how does the number of bits in each entry of the page table change?

1 bit smaller. half as many physical pages

Note: phys page size == virtual page size

If the page size (in bytes) is doubled, how does the number of entries in the page map change?

half as many entries, half as many virtual pages

The following table shows the first 8 entries in the page map. Recall that the valid bit is 1 if the page is resident in physical memory and 0 if the page is on disk or hasn't been allocated.

	Valid Bit	Physical Page
0	0	7
1	1	9
2	0	3
3	1	2
4	1	5
5	0	5
6	0	4
7	1	1

10 bits of offset

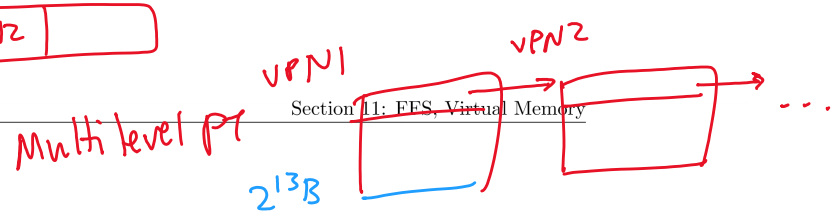
If there are 1024 bytes per page, what is the physical address corresponding to the hexadecimal virtual address 0xF74?

PPN = 2 ⇒ addr = 0b10 1101 110100

0b1111 0111 0100

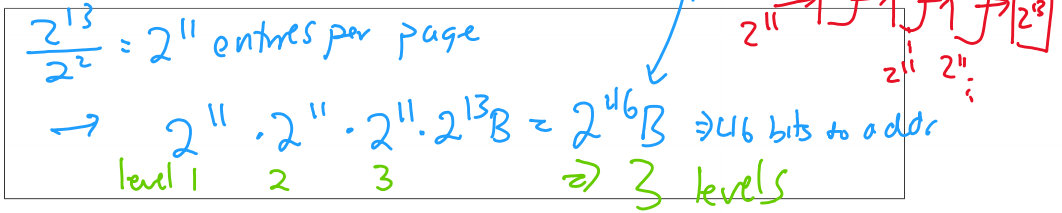
VPN = 0b11 = 3

= 0xB74

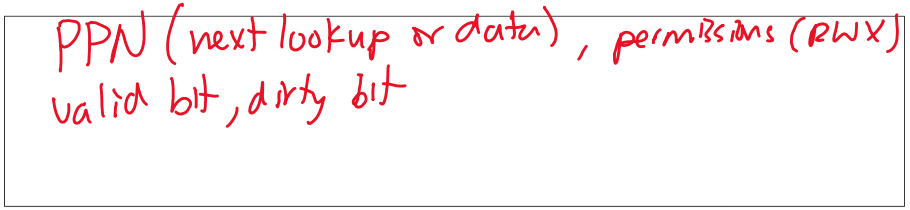


3.2 Address Translation

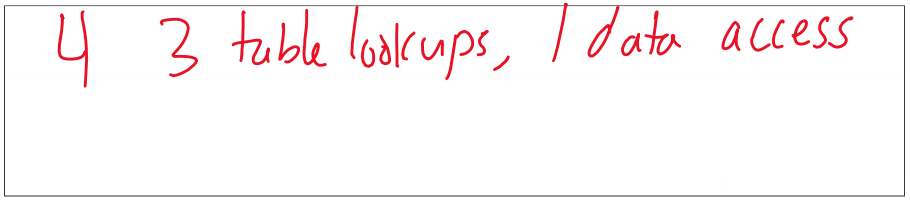
Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page?



List the fields of a Page Table Entry (PTE) in your scheme.



Without a cache or TLB, how many memory operations are required to read or write a single 32-bit word?



With a TLB, how many memory operations can this be reduced to? Best-case scenario? Worst-case scenario?